GITLAB-CI

Cette présentation contient les bases et les bonnes pratiques pour utiliser Gitlab-CI

Université Gustave Eiffel

Thibault SAUSSAC - saussact@gmail.com

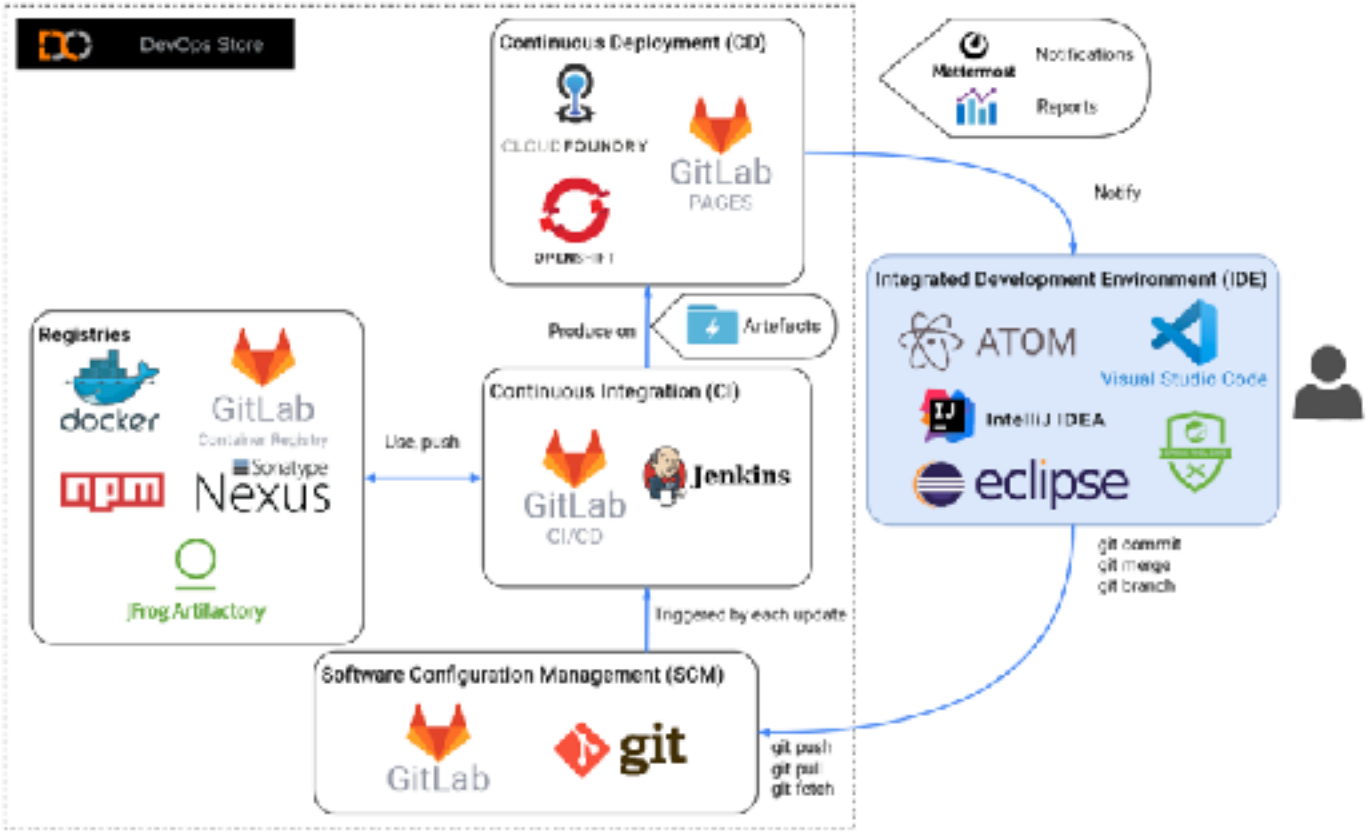
Les enjeux du DevOps

- **de réduire le Time To Market** :
  - délai entre l'idée (fonctionnalité, correction de bug) et sa mise à disposition pour le client
  - arrivée sur le marché plus rapide
  - réduction du stock de code qui végète avant qu'il soit mis en production

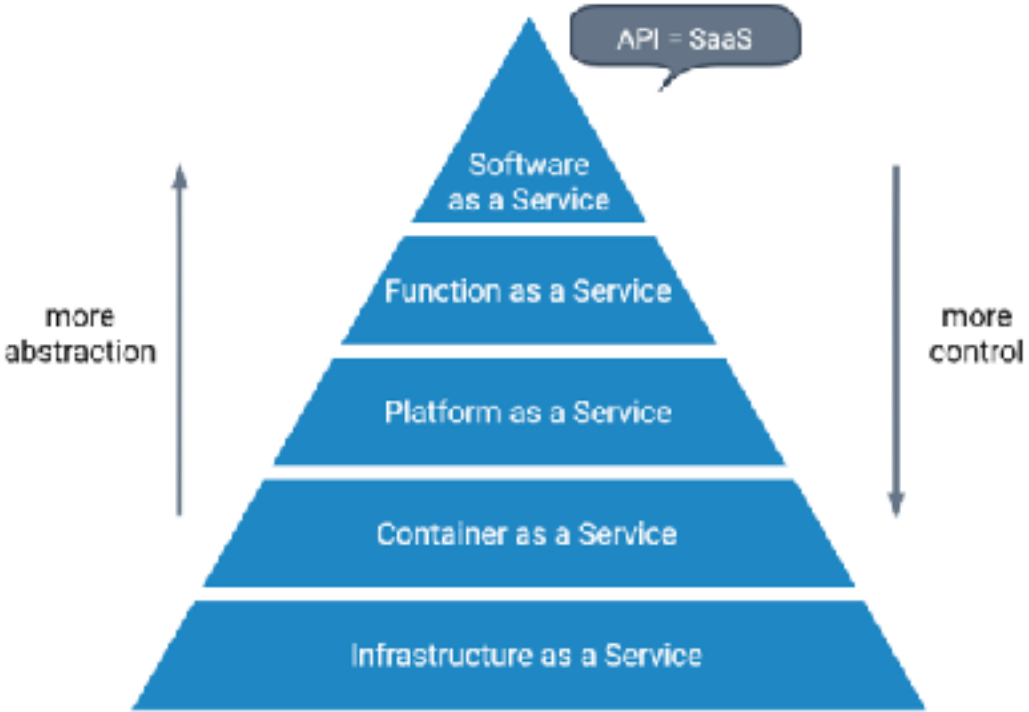
- **d'apprendre des retours clients et des métriques** :
  - rectifier le tir si besoin et au plus vite
  - éviter le coût important de réorientation/recentrage.

- **de diminuer les risques et le stress** :
  - la mise en production devient une activité quotidienne et maîtrisée
  - mise en production de petits morceaux de code bien identifiés
  - ciblage rapide du code défectueux, avec retour en arrière facilité
  - prise en charge des demandes (évolutions, corrections...) au fil de l'eau
  - mises en production tous les 6 mois : les demandes affluent pour ne pas manquer ce train

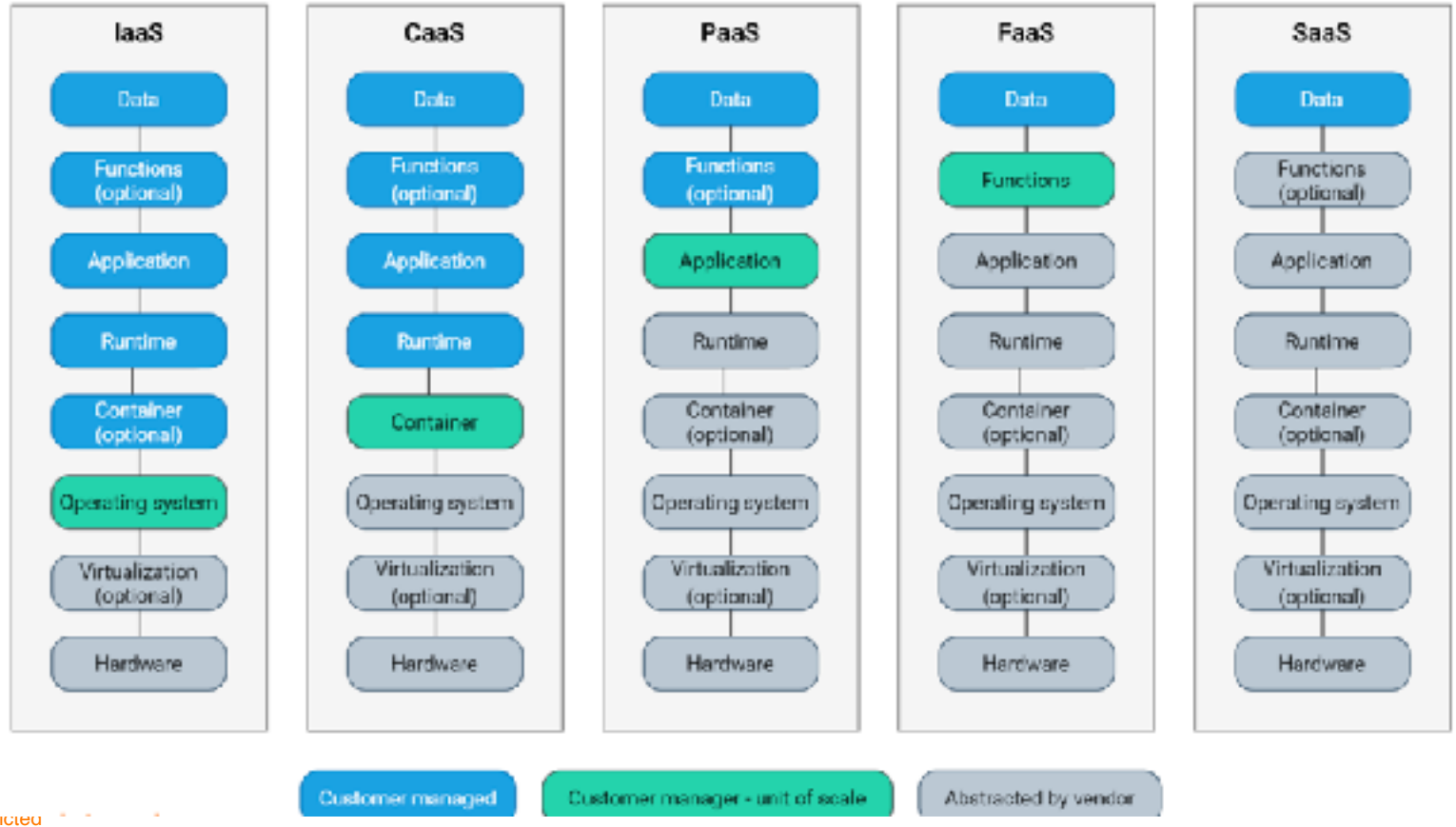
Ecosystème



Les XaaS - Everything As a Service



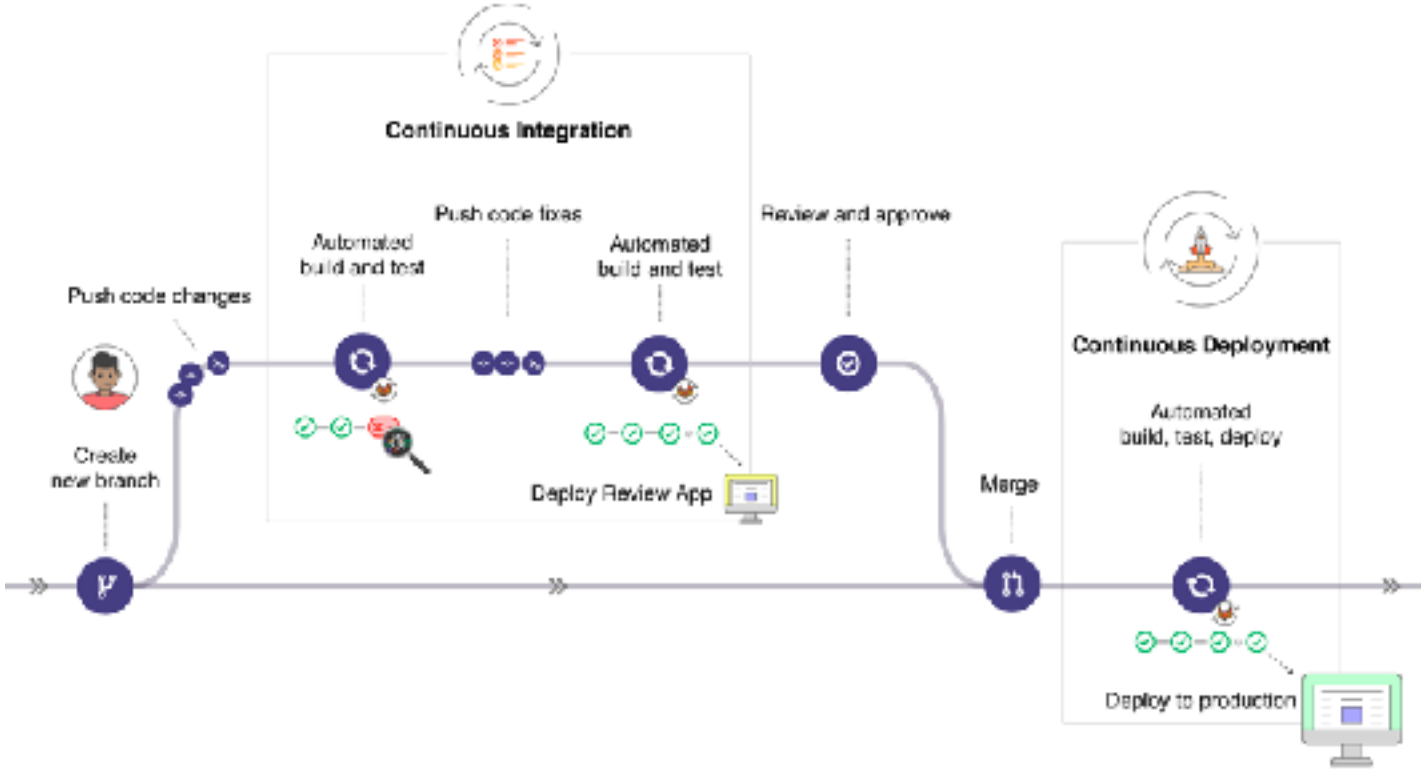
Les XaaS - Everything As a Service



La CI / CD, c’est quoi ?

- Les méthodes qualifiées de **`continue`** de développement logiciel sont basées sur **l'automatisation** via l'éxécution de scripts.
  - Minimiser les chances d'introduction d'erreur lors du développement d'application.
  - Moins d'interventions humaines.

La chaine de CI / CD dans Gitlab



CI - Continuous Integration

L'intégration d'une application avec la mise en oeuvre de pré-requis est un exemple d'intégration continue:

- Intégration des scripts d'installation de l’application
- Ajout de scripts spécifiques pour créer un réseau, des clefs OpenStack etc (ex Terraform)
- Création de tests (Junit par exemple)
- Les tests associés à ce code peuvent être des checks de linting, de sécurité
- Ce premier déploiement est fait sur un environnement d’intégration

Définition imagée : C'est l'élaboration de la recette du gateau

CD - Continuous Delivery ou Continuous Deploiement

Continuous Delivery: Livraison continue

Si on reprend l'exemple précédent d'intégration d'une VNF vendor, on ajoute une phase de déploiement sur un environnement de pré-production (staging)

Définition imagée: La recette du gateau est publiée

Continuous Deployment: Déploiement continue

On ajoute le déploiement avec déclenchement manuelle ou automatique sur l'environnement de production

Définition imagée: C'est la fabrication réelle du gateau

Il y a toujours des steps de build (Utilisation de fichiers d'entrée pour l'intégration avec l'environnement cible) et de tests

Quelques services …

- Un wiki
- un suivi d’issue,
- un registry docker,
- un suivi de code,
- une review de code
- une CI/CD,
- …

Pipeline comme support des phases de CI/CD

Les différentes étapes de CI/CD sont executées dans des pipelines.

Un pipeline est constitué de jobs qui sont organisés et ordonnancés par des stages. Sous gitlab, le pipeline est défini par un fichier `.gitlab-ci.yml : « Le manifest »`

Définition des : stages, jobs, variables… etc

Les Jobs

Vous pouvez définir un nombre illimité de `jobs (avec nom unique hors… images, service, types… etc)`

```
job:
  script: echo 'my 1st job'
job2:
  script: echo 'my 2nd job'
```

script

C’est la seule section obligatoire dans un job… c’est la qu’on écrit les commandes à lancer.

```
job1:
  script: ./script/my-script.sh ## lance un script du projet

job2:
  script: ## lance 2 script du projet
    - ./script/my-script-1.sh
    - ./script/my-script-2.sh

jobcommand2:
  script: # Exécution de deux commandes
    - printenv
    - echo $USER
```

before_script et after_script

Permettront d'exécuter des actions avant et après votre script principal.

```yaml
before_script: # Exécution avant chaque `job`
  - echo 'start jobs'

after_script: # Exécution après chaque `job`
  - echo 'end jobs'

job:no_overwrite: # `before_script` et `after_script` par défaut
  script:
    - echo 'script'

job:before_script:
  before_script:
    - echo 'before_script' # N'exécutera pas le `before_script` par défaut
  script:
    - echo 'script'

job:after_script:
  script:
    - echo 'script'
  after_script:
    - echo 'after_script' # N'exécutera pas le `after_script` par défaut
```

Images

Cette déclaration est simplement l'image docker qui sera utilisée lors d'un job ou lors de tous les jobs

```
image: alpine #ce sera l'image par défaut
```

```
job:node:
  image: node
  script: yarn install
```

Les stages

```yaml
stages: # déclarations des étapes
  - build
  - test
  - deploy

job:build:
  stage: build
  script: make build

job:test:unit:
  stage: test
  script: make test-unit

job:test:functional:
  stage: test
  script: make test-functional

job:deploy:
  stage: deploy
  script: make deploy
```

Démo.